11/09/2023
Monday

* 2-D ARRAYS →

A two-dimensional structure containing rows and columns is known as 2-D Array.

* DECLARATION OF 2-D ARRAY →

```
int arr [rows][columns];
```

For instance,

```
int arr [5][7];
char ch [3][2];
```

* INITIALIZATION OF 2-D ARRAY →

```
int arr[3][4] = {
                  {1, 2, 3, 4},
                  {5, 6, 7, 8},
                  {9, 10, 11, 12}
                };

char ch [3][2] = {
                  {'A', 'B'},
                  {'C', 'D'},
                  {'E', 'F'}
                };
```

* ACCESS OF ELEMENTS →

```
int arr[2][2] = {
                  {10, 20},
                  {30, 40}
                };
```

arr[i][J]

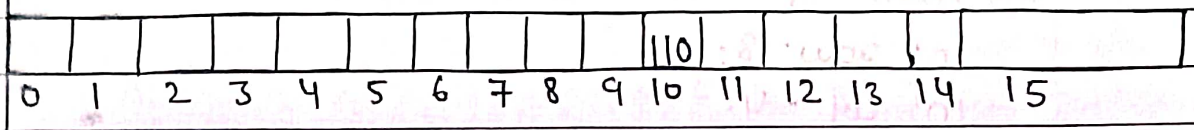
where i = row index
J = column index

Behind the scenes →

There is no allocation of 2-D space in memory for 2-D arrays. They are just stored like linear array in continuous memory blocks.

For instance,

	0	1	2	3
0	10	20	30	40
1	50	60	70	80
2	90	100	110	120
3	130	140	150	160



Mapping of 2-D array in memory is done by using formula →

c x i + J

where c = columns (total), i = row index, J = column index

Eg - 110 is at (2,2)

In memory, 4 x 2 + 2 = 8 + 2 = 10th index is allocated for 110.

NOTE- When an array i.e. 2-D is initialized, it is mandatory to give the size of columns because mapping of 2-D array in linear array requires size of columns.
Also, when a 2-D array is passed, the size of columns must be told.
It's optional to give the size of rows in 2-D array.

* **Program to print 2-D Array →**

```
#include<iostream>
using namespace std;

void printArray (int arr[][4], int row, int col){
    for (int i=0; i<row ; i++){
        for (int j=0; j<col; j++){
            cout << arr[i][j]<<" ";
        }
        cout<< endl;
    }
}
int main(){
    int row=3;
    int col =4;
    int arr[][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};

    printArray (arr, row, col);
}
```

Output →

```
1  2  3  4
5  6  7  8
9  10 11 12
```

* NOTE → The below program will work for both transpose of square and non-square matrix.

__/__/__

* Program to print 2-D array (column-wise) →

I/P = 1 2 3 4
5 6 7 8
9 10 11 12

O/P = 1 5 9
2 6 10
3 7 11
4 8 12

```
#include<iostream>
using namespace std;

void printArray(int arr[][4], int row, int col){
    for(int i=0; i<col; i++){
        for(int j=0; j<row; j++){
            cout<<arr[j][i]<<" ";
        }
        cout<<endl;
    }

int main(){
    int row = 3;
    int col = 4;
    int arr[][4]={
                {1,2,3,4},
                {5,6,7,8},
                {9,10,11,12}
                };

    printArray(arr, row, col);

}
```

__/__/____

* **Program to take input elements in 2-D array and printing them →**

```
#include <iostream>
using namespace std;

int printArray (int arr[2][3], int row, int col){
    for (int i=0; i<row; i++){
        for (int j=0; j<col; j++){
        cout<<"row index: "<<i<<" "<<"column index: "<<j;
        cin >> arr[i][j];
        }
    }
}
int main (){
    int row = 2;
    int col = 3;
    int arr[2][3];

    printArray (arr, row, col);

    for (int i=0; i<row; i++){
        for (int j=0; j<col; j++){
            cout<< arr[i][j] <<" ";
        }
        cout<<endl;
    }
}
```

Input given →

row index : 0	column index : 0	10
row index : 0	column index : 1	20
row index : 0	column index : 2	30
row index : 1	column index : 0	40
row index : 1	column index : 1	50
row index : 1	column index : 2	60

Output → 10 20 30
40 50 60

* **Program for linear search in 2-D array →**

I/P =

23	45	67
12	56	97
43	50	41

} 2-D Array

Target = 12

O/P = Target found or not.

Logic → Traverse the loop for rows and columns.
Check if target = arr[i][j], then →
return 1

else
return 0

__/__/____

```
#include <iostream>
using namespace std;

bool searchTarget (int arr[3][3], int row, int col,
                   int target){

    for (int i=0; i<row ; i++) {
        for (int J=0; J<col ; J++){
            if (arr[i][J] == target){
                return 1;
            }
        }
    }
    return 0;
}

int main(){
    int row = 3;
    int col = 3;
    int target= 12;
    int arr[3][3] = {
                        {23, 45, 67},
                        {12, 56, 97},
                        {43, 50, 41}
                    };
    cout<<"target found or not: "<<
                searchTarget(arr, row, col, target);

}
```

__/__/____

* Program to print maximum in 2-D Array →

I/P = 45 5 670
23 80 4000
11 78 8
} 2-D array

O/P = maximum is: 4000

Logic → Traverse the loop for rows and columns.
Initialize int max = INT_MIN;
if (arr[i][j]) > max
max = arr[i][j]
return max

Program

```
#include<iostream>
using namespace std;
#include<limits.h>
int printMax(int arr[3][3], int row, int col){
    int max = INT_MIN;
    for(int i=0; i<row; i++){
        for(int j=0; j<col; j++){
            if(arr[i][j] > max){
                max = arr[i][j];
            }
        }
    }
return max;
}
int main(){
int arr[3][3] = {{45,5,670},{23,80,4000},{11,78,8}};
int row = 3; int col = 3;
int finalAns = printMax(arr, row, col);
cout<<"maximum is: "<<finalAns<<endl;
}
```

__/__/____

* Program to print row-wise sum →

I/P = 1 2 3
4 5 6
7 8 9
10 11 12

O/P = sum for row 0 is : 6
sum for row 1 is : 15
sum for row 2 is : 24
sum for row 3 is : 33

Logic → Traverse the loop for row, initialize the sum with 0.
Traverse the 2nd loop for columns.
Sum = sum + arr[i][J]
Print sum

Program

```
#include<iostream>
using namespace std;
void printRowSum(int arr[4][3], int row, int col){
    for(int i=0; i<row; i++){
        int sum=0;
        for(int J=0; J<col; J++){
            sum= sum + arr[i][J];
        }
cout<<"sum for "<<"row"<<i<<" is : "<<sum<<endl;
    }
}.
int main(){
 int arr[4][3] ={{1,2,3},{4,5,6},{7,8,9},{10,11,12}};
 int row = 4;
 int col = 3;

 printRowSum(arr, row, col);
}
```

* for column-wise sum →

I/P = 1 2 3
4 5 6
7 8 9
10 11 12

O/P = Sum for col 0 is : 22
sum for col 1 is : 26
sum for col 3 is : 30

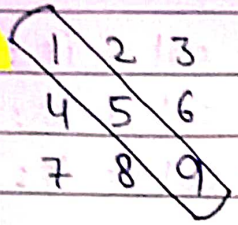
Logic → Traverse the outer loop for column. Initialize the sum with 0. Traverse the inner loop for rows. sum = sum + arr[J][i]
Print sum

Program

```
#include<iostream>
using namespace std;
void colSum(int arr[4][3], int row, int col){
    for(int i=0; i<col; i++){
        int sum=0;
        for(int J=0; J<row; J++){
            sum= sum+ arr[J][i];
        }
cout<<"sum for col"<<i<<" is: "<<sum<<endl;
    }
}
int main(){
  int arr[4][3]={{1,2,3},{4,5,6},{7,8,9},{10,11,12}};
  int row=4;
  int col = 3;

  colSum(arr, row, col);
}
```

* Program to print sum of diagonal in 2-D array →

I/P = 

O/P = Diagonal sum: 15

logic → Diagonal is formed only when no. of rows and columns is equal. As no. of rows = no. of columns, so, we have to traverse only one loop either for row or column. let for row →

```
Sum = Sum + arr[i][i]
Print sum
```

Program

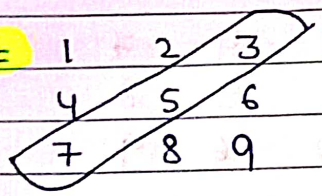
```
#include<iostream>
using namespace std;
void DiagonalSum(int arr[3][3], int row, int col){
    int sum = 0;
    for(int i=0; i<row; i++){
        sum = sum + arr[i][i];
    }
 cout<< "Diagonal sum: "<< sum<<endl;
}
int main(){
   int row = 3;
   int col = 3;
   int arr[3][3] = {{1,2,3},{4,5,6},{7,8,9}};

   DiagonalSum(arr, row, col);

}
```

* **Program to print sum of reverse diagonal →**

I/P =



O/P = Diagonal sum: 15

Logic → Initialize sum with 0. Take n=3 (n represents no. of rows & columns). Traverse two loops for row & column.

```
if (J==n-i-1)
    sum = sum+arr[i][J]
    Print sum
```

Program

```
#include <iostream>
using namespace std;
void DiagonalSum (int arr[3][3], int row, int col){
 int sum = 0;
 int n= 3; // n represents total rows and cols
 for (int i=0; i<row; i++){
   for (int J=0; J<col; J++){
      if (J==n-i-1){
         sum=sum+ arr[i][J];
       } } }
  cout<<"reverse diagonal sum: "<< sum <<endl;
}
int main(){
  int row = 3;
  int col = 3;
  int arr[3][3]={{1,2,3},{4,5,6},{7,8,9},{10,11,12}};

  DiagonalSum(arr, row, col);
}
```

__/__/____

* Program to print transpose of a square matrix →

I/P = 1 2 3
4 5 6
7 8 9

O/P = 1 4 7
2 5 8
3 6 9

Logic → Traverse outer loop for i=0, i<row
Traverse inner loop for J=0, J<i+1
Swap = arr[i][J] ⟷ arr[J][i]
Print it.

Program

```
#include <iostream>
using namespace std;
void transpose(int arr[3][3], int row, int col){
  for(int i=0; i<row; i++){
    for(int J=0; J<i+1; J++){
      swap(arr[i][J], arr[J][i]);
    }}
  for(int i=0; i<row; i++){
    for(int J=0; J<col; J++){
      cout<<arr[i][J]<<" ";
    }
    cout<<endl;
  }
int main(){
  int row = 3;
  int col = 3;
  int arr[3][3]={{1,2,3},{4,5,6},{7,8,9}};
  transpose(arr, row, col);
}
```

* 2-D VECTOR →

- **Declaration →**

```
vector <vector<int >> v;
```

- **Initialization →**

=
```
vector <vector<int>>v {
                        {1,2,3},
                        {4,5,6}
                      };
```

=
```
vector <vector<int >>v = {
                           {1,2,3},
                           {4,5,6}
                         };
```

=
```
vector<vector<int>>v (5, vector<int>(5,2));
```

number of rows ← 5 , vector<int>(5,2) → what happens in every row

If we print this statement, the output will be →

```
2 2 2 2 2
2 2 2 2 2
2 2 2 2 2
2 2 2 2 2
2 2 2 2 2
```

__/__/____

* **Program to print a 2-D vector →**

```
#include <iostream>
# include <vector>
using namespace std;

int main( ){
   vector < Vector <int >> v {
                              {1,2,3},
                              {4,5,6}
                             };

for (int i=0 ; i< v.size( ) ; i++){
   for (int J=0; J< v[i].size( ) ; J++){
         cout << v[i][J] <<" ";
      }
     cout <<endl;
   }
}
```

Output =

```
1 2 3
4 5 6
```

v.size( ) → size of rows
v[i].size( ) → size of column

* **Jagged Array →** Jagged array is that in which number of columns may vary. It can be done by making a 2-D vector and push_back 1-D vectors inside that 2-D vector.

```
#include <iostream>
#include < vector >
using namespace std;

int main() {
    vector <vector <int >> v;  // 2-D vector

    vector <int> v1 (1, 1);
    vector <int> v2 (2, 2);    // 1-D vector
    vector <int> v3 (3, 3);

    v. push_back (v1);
    v. push_back (v2);         //push_back 1-D vector in 2-D vector
    v. push_back (v3);

    for (int i=0; i<v.size(); i++){
      for (int J=0; J< v[i]. size(); J++){
          cout<< v[i][J]<<" ";
        }
      cout<<endl;
    }
}
```

Output →

```
1
2  2
3  3  3
```